Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/5540--02-8635

A Detection Study of an NRL Steganographic Method

Ira S. Moskowitz

Center for High Assurance Computer Systems
Information Technology Division

Neil F. Johnson
Michael Jacobs

Center for Secure Information Systems
George Mason University
Fairfax, VA

August 16, 2002

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE August 16, 2002	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE

A Detection Study of an NRL Steganographic Method

5. FUNDING NUMBERS

WU - 55-8189-02

6. AUTHOR(S)

Ira S. Moskowitz, Neil F. Johnson,*† and Michael Jacobs†

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5540--02-8635

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

*Currently with Booz Allen Hamilton National Security Team, McLean, VA
†Center for Secure Information Systems, George Mason University, Fairfax, VA 22030

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

In this report we analyze in detail a method of image steganography developed by NRL. Our conclusion is that this method of steganography is undetectable by current pragmatic statistical stego detection techniques, primarily because it alters a very small number of pixels. The small size of the embedded message is the key to the lack of detection, provided that a non-anomalous cover image is used.

14. SUBJECT TERMS

Information hiding	Steganalysis
Steganography	TIFF

15. NUMBER OF PAGES

28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CONTENTS

A Detection Study of an NRL Steganographic Method

1 Introduction

We report on the steganalysis of a stego algorithm developed by the Naval Research Laboratory (NRL), called the NRL method. The steganalysis was done by the Center for Secure Information Systems (CSIS) at George Mason University (GMU). NRL provided CSIS with images, which they then analyzed. CSIS was not able to determine if an image was a "clean" cover image, or a "suspect" stego image. The NRL method is a variant of the well-known least significant bit (LSB) hiding technique [3]. The NRL method is not robust against compression, nor in its elementary form is it robust against noise. The NRL method is used to demonstrate that a small enough embedded message can be hidden, without the steganography being detected.

The steganalysis performed by CSIS is based on investigative techniques being developed at CSIS for targeting open-source and commercial steganography software tools and algorithms. This task involves investigating techniques employed for steganography and the resulting stego media to determine if any *stego signatures* can be readily detected.[1] We also use this report to showcase some statistical stego detection techniques developed by CSIS.

2 Steganography

We give a brief description of still image-based steganography. The **cover image** is the image in which we do the hiding. The **embedded message** is what we hide, via the (steganographic or) stego algorithm or method. Note that the embedded message need not be an image. In fact, in the NRL method the message is ASCII text (including the Unix line feed "0a"). The stego image is the cover image with the embedded image "in it." The **extracted message** is what we extract from the stego image by reversing the stego algorithm. The extracted message need not be exactly the same as the embedded image. The acceptable quality level depends on the usage and data type [5]. For the NRL method, no errors are tolerated (the NRL method can be modified, by using error correcting codes, to deal with loss of fidelity). Also, the stego method may or may not use a "key," referred to as the **stego key**.

The NRL method discussed here uses a stego key. For key based steganography, the steganography should not be detectable without the key. Recall that in steganography the thrust is to keep hidden the very *existence* of the embedded message. Once an eavesdropper knows that there is a hidden message, the steganography has failed. Of course, by only embedding encrypted messages, one could make the hidden message unintelligible to the eavesdropper. However, that is an issue of cryptography --- the steganography is still considered to have failed.

[1] *Stego signature* – Steganography media signatures include characteristics or patterns introduced to carrier media based on the impact of using steganography techniques to hide information in digital media.

Manuscript approved July 19, 2002.

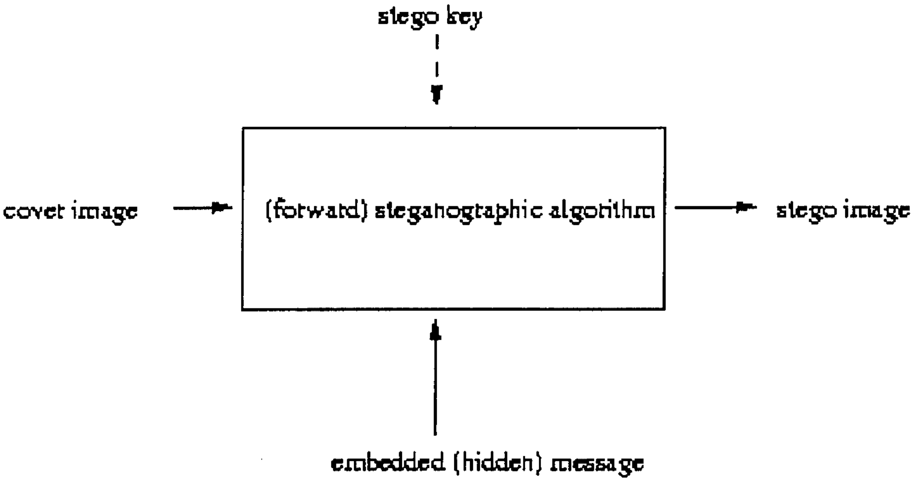


Figure 2.1: Forward Stego Algorithm

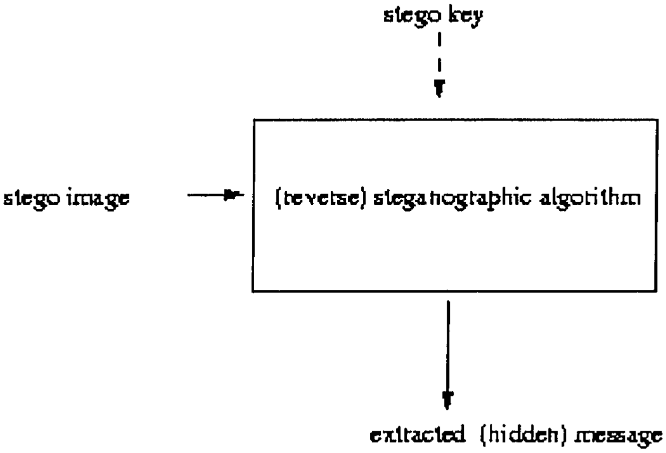

stego key
stego image
(tevetse) steganogtaphic algotithm
extracted (hidden) message

Figure 2.2: Reverse Stego Algorithm

We stress that a stego key is not necessary. This may seem to go against the nature of Kerchoffs' principle [1], which is a mainstay of cryptography. If one wishes to determine whether a given image is a stego image or not, and they have knowledge of the stego algorithm, then the determination is trivial. However, steganography differs from cryptography in that we might be attempting to determine which images are stego images from a set of millions of images (such as image postings on the Web, or on USENET BBDs). In that case, even with knowledge of the stego algorithm, the task is still daunting.

2.1 Kurak & McHugh

One of the simplest methods of image-based steganography is that of swapping the least significant bits (LSBs) of the cover image, with the most significant bits (MSBs) of the embedded message. This method is discussed in detail in [3].

2.2 NRL Method

The Kurak & McHugh method of steganography is easily detectable because entire bit planes have been swapped. The NRL method hides much less data, but is also much less detectable. The NRL method hides a maximum of 249 ASCII characters in either a greyscale or a color image (a spatially realized TIFF file) that have minimum dimensions of 500 x 500. A stego key is randomly generated of length 1000. The key is made up of 1000 unique tuples, where each tuple is of the form (x_i,y_j) where both x_i and y_j are integers in the range 0 to 499. (Thus, one sees why both the horizontal and vertical dimensions of the image must be at least 500.) The stego key gives us 1000 image coordinates in terms of pixel values. We used the "xv" software for our image processing. For an M x N image in xv, the x values run from left to right (0 to M-1), and the y values run from top to bottom (0 to N-1). We only deal with a spatially realized image, so we are using the bitmap representation. We save our files in uncompressed TIFF format.

Given an ASCII character we break it up into four 2-bit pairs as follows: if the ASCII characters byte representation is $(b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8)$, where each b_i is a bit, we break it up into (b_1,b_2), (b_3,b_4), (b_5,b_6), (b_7,b_8). Our embedded message has a maximum length of 249. We always append a stop signal to the information carrying part of the message. The stop signal is the four 2-bit pairs (0,0), (0,0), (0,0), (0,0). In other words, the stop sequence is the four 2-bit pair decomposition of the null symbol. A pixel coordinate (x_i,y_j) has an RGB value (Ri,j ,Bi,j ,Gi,j). Of course if the image is greyscale all three colors have the same value. Each color value is given by a byte (i.e., an integer 0 to 255).

We concentrate our effort on the red value Ri,j. We swap the two least significant bits (LSBs) of Ri,j with a two bit pair from an ASCII character if the image is color, and we swap every color's two LSBs if the image is greyscale. The stego key tells us what order to change the pixels in for the stego embedding, and how to read them out for the stego extraction. We treat color and greyscale differently because for a greyscale the three colors must all have the same value or else the steganography will be obvious. For a color image we only change the red value to minimize the changes to the cover image with the intent of making the steganography as hidden as possible. The algorithm is as follows.

1-Take an ordered ASCII text message up to 249 characters.

2-Break each character up into four ordered 2-bit pairs as discussed above. Thus, we now have an ordered message of up to 996 2-bit pairs.

3-For the first tuple in the stego key replace the two LSBs of the red (all colors) byte of that pixel with the first 2-bit pair from the ordered message if the image is color (greyscale). For the second tuple in the stego key, replace the two LSBs of the corresponding pixel byte value(s) with the next two bit pair, the same for the third and fourth tuples from the stego key. At this point we have embedded the first ASCII character into the cover image. If there is a second ASCII character we use the fifth tuple from the stego key. We keep doing this until all the *n* ASCII (n <250) characters have been embedded into the cover image.

4-Once this is done we use the 4*n*+1, 4*n*+2, 4*n*+3, and 4*n*+4 tuple coordinates to embed the null sequence into the cover image in the same manner as above.

We have now replaced the cover image with the stego image. The stego image has at most a difference of 1000 pixel values with that of the cover image. That is no more than 0.4% of the pixels have been changed, and if the pixel has been changed it is only in the 2 LSBs. The differences are only in the 2 LSBs of the red byte (all color's bytes) value if the image is color (greyscale). These changes are not noticeable to the human visual system (HVS) for a "good" cover image.[2] For any steganographic method one should only use "good" cover images.

The stego extraction simply reverses the processes, extracting each ASCII character out in four 2-bit pairs. The stego key indicates which pixels to compose the characters from and in what order to do it. When the extraction process reaches the null character the extraction is complete and the extracted message is the same as the embedded message.

We contend that it is extremely difficult to detect this NRL method of steganography. The remainder of this report backs that claim.

3 Methodology

This section describes the approach taken in reviewing the media provided by NRL by CSIS.

3.1 Assumptions and Constraints

A study of the steganography described above was done to determine if the NRL method has characteristics that produce detectable signatures. The following assumptions and constraints apply:

- The steganography tool used to create the stego image was not made available to CSIS for analysis or to produce known cover/stego media pairs for analysis. Therefore, the analysis performed by CSIS was limited to the images (the carrier images) provided by NRL for investigation. CSIS was tasked to determine which images were in fact stego images and which were benign cover images.
- However, after the initial testing the method (not the tool itself) was revealed to CSIS, but the NRL method could still not be detected. Of course CSIS was not given cover image-stego image pairs. In other words CSIS was tasked to perform blind detection. Neither was CSIS given the stego keys. This would be the best that an informed eavesdropper could hope for.

[2] A "bad" or anomalous cover image would be one, for example, that is all black.

3.2 Detecting a Steganography Signature---General Approach

Identifying *steganography signatures* is based on assessing combinations of carrier, stego media, embedded message, and steganography tools known by the analyst. One must look for characteristics or patterns based on the impact the steganography tools have on the digital media used to hide information. These characteristics may be used to identify the use of steganography in a system and to identify the modified media. This portion of the analysis involves identifying recurring characteristics or patterns produced by the steganography software and is one of the most time consuming activities.

The steps typically used by CSIS to analyze existing steganography tools includes:

1. Obtaining a collection of carriers without information embedded within them (we will call this the set of *original* carriers).[3]
2. Hiding information in the various carriers using a steganography tool to produce *stego media* files (this is the set of original carriers that now contain hidden information).
3. Performing a number of tests against the stego media to determine if a pattern can be detected.

Since the tools used to hide information may not be available, step 2 above may not be viable. The objective of analyzing the stego media is to find a pattern that can be detected without requiring an original for inspection.[4] At this point our options for determining signatures involve visual inspection and determining if statistical anomalies exist.

Identifying Steganography Signature

Identifying *steganography signatures* is based on assessing combinations of carrier, stego media, embedded message, and steganography tools known by the analyst. In this experiment, only the stego media is provided for evaluation. Discovering signatures in stego media relies on other understanding the analysts have with respect to tools and techniques for hiding information as well as the associated stego signatures.

In making comparisons with numerous images, patterns begin to emerge as possible signatures of steganography software. Some of these signatures may be exploited automatically to identify the existence of hidden messages and even the tools used to embed the messages. With this knowledge base, if the cover images are not available for comparison (as in the NRL tests), the derived known signatures are enough to imply the existence of a message and identify the tool used to embed the message.

The approaches used in analyzing the images provided by NRL are visual inspection and statistical analysis. In some cases these techniques can be employed to identify the tools used to hide information, in other cases, these detection techniques may simply be used to identify suspicious characteristics. The following 16 images were provided by NRL.

[3] For the most part the carriers focused on in the past were images.

[4] Also known as *blind detection*.

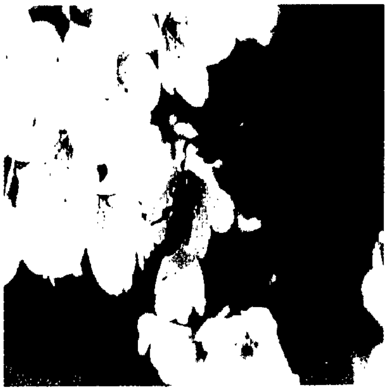


Figure 3.1: test1.tif

Figure 3.2: test2.tif

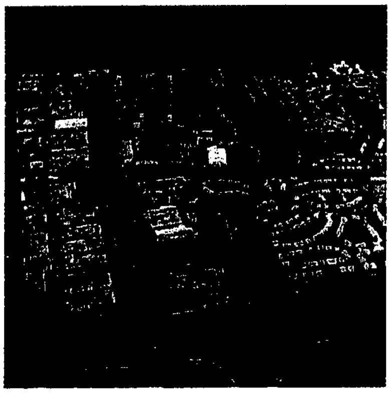


Figure 3.3: test3.tif

Image not shown

Figure 3.4: test4.tif

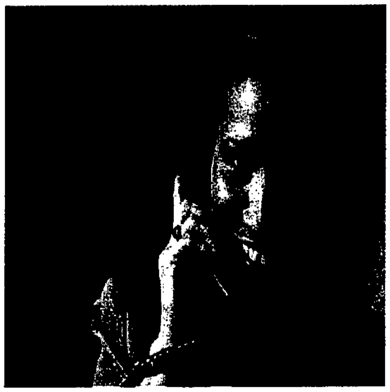


Figure 3.5: test5.tif

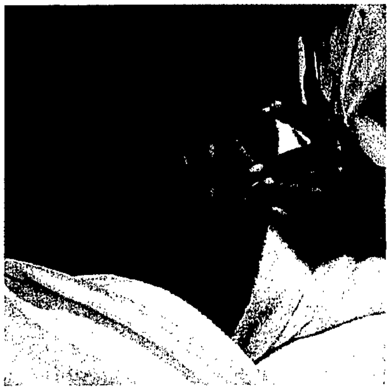


Figure 3.6: test6.tif

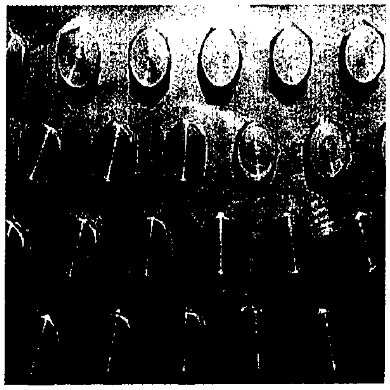


Figure 3.7: test7.tif

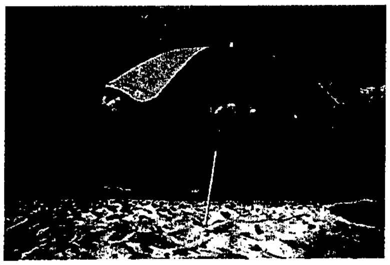


Figure 3.8: test8.tif

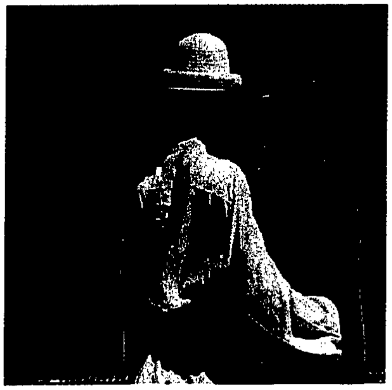


Figure 3.9: test9.tif

Figure 3.10: test10.tif

Figure 3.11: test11.tif

Image not shown

Figure 3.12: test12.tif

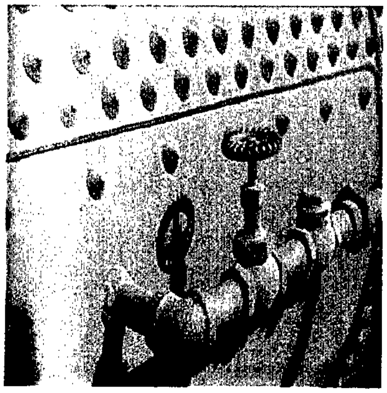


Figure 3.13: test13.tif

Figure 3.14: test14.tif

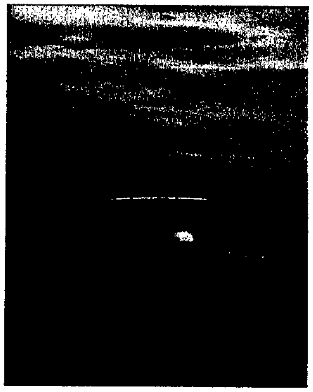


Figure 3.15: test15.tif

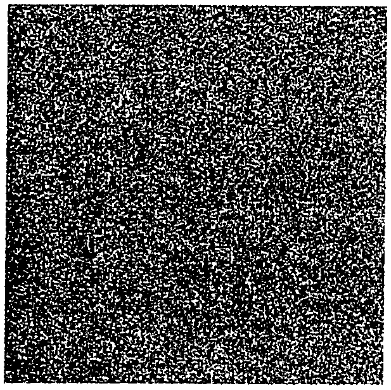


Figure 3.16: test16.tif

NRL Images

The above sixteen TIFF format images in Figure 3.1 through Figure 3.16 were provided by NRL for steganalysis. These images are 24-bit TIFF images with dimensions 500 pixels in width by 500 pixels in height with the following exceptions:

test8.tif – 792 pixels in width by 528 pixels in height
test11.tif – 720 pixels in width by 576 pixels in height
test12.tif – 776 pixels in width by 552 pixels in height
test15.tif – 576 pixels in width by 720 pixels in height

4 TIFF Files

The images files provided by NRL were all TIFF. No compression was used when saving the images in TIFF format to assure that the lower bits were not over written. This section describes the structural elements of a TIFF file using the image in Figure 3.2 as an example. In this section we will be looking at the digital and hexadecimal representation of the image in Figure 3.2: test2.tif.

Figure 3.2: test2.tif

4.1 Structure of a TIFF File

A TIFF file always begins with either II*[NULL] or MM[NULL]* followed by a 32-bit value that is the offset position for the image file directory (IFD), which contains information about the image such as height, width, depth, number of color planes, and type of data compression.[5] No compression was used in the test images. The header for the data in Figure 4.17 begins at offset 00 and ends with the byte at offset 07. The IFD offset position is identified as 0B71B8 by the last four bytes of the file header.

[5] Though translated ASCII data differs for some characters between UNIX, Windows, and DOS environments, the Hexadecimal Data (or actual binary values) remain the same. The ASCII data presented in the figures illustrating the HEX Dump data of the TIFF files are for DOS systems.

```
OFFSET       Hexadecimal Data                                            ASCII Data
00000000:    4D 4D 00 2A 00 0B 71 B8 | 6F 67 57 53 4F 3F 5B 5B         MM * ♂q╕ogWSO?[[
00000010:    4B 47 3B 37 53 4B 3F 4B | 3F 3F 4F 4F 4B 63 5F 5B         KG;7SK?K??OOKc_[
00000020:    6B 63 5F 6B 6F 6B 7F 87 | 83 7F 83 7B 77 7B 7B 77         kc_kok⌂ç⌂â{w{{w
00000030:    77 6F 73 6F 6B 6B 5B 53 | 6B 57 53 6F 67 57 67 57         wosokk[SkWSogWgW
00000040:    3B 73 4F 3F 63 57 47 6B | 5F 57 77 73 6B 77 6F 67         ;sO?cWGk_Wwskwog
00000050:    8B 87 77 8B 7F 6B 6B 73 | 5B 7F 67 57 6F 6F 57 6B         ïçwï⌂kks[⌂gWooWk
00000060:    5B 4F 6B 67 5B 8B 97 7F | 3F 43 3F 6B 5B 5F 7F 67         [Okg[ïù⌂?C?k[_⌂g
00000070:    6B 7F 5B 5B 93 73 47 97 | 7B 47 6F 6F 4A 6B 67 5E         k⌂[[ôsGù{GooJkg^
00000080:    5F 56 52 67 66 6A 5B 4F | 56 6B 5F 63 5B 46 5A 6A         _VRgfj[OVk_c[FZj
00000090:    61 65 49 5D 51 35 39 41 | 61 60 5C 41 40 3C 35 34         aeI]Q59Aa`\A@<54
000000A0:    40 3D 38 38 49 3C 39 4D | 34 38 49 38 38 3D 3C 38         @=88I<9M48I88=<8
000000B0:    45 40 34 45 40 38 45 40 | 3C 45 44 38 41 3C 38 45         E@4E@8E@<ED8A<8E
```

Figure 4.17: Hex Dump of test2.tif (first 192 bytes)

The IFD begins at offset 0B71B8 (or at the 750,008th byte of the file). In the image test2.tif (Figure 3.2) the IFD follows the image *raster data* (*raster data* is "what you see" on the screen. It is the data values that actually produce the visible image).

```
Offset       Hexadecimal Data                                            ASCII Data
000B71B0:    43 3F 63 47 43 5B 4F 47 | 00 0F 01 00 00 03 00 00         C?cGC[OG ☼☺  ♥
000B71C0:    00 01 01 F4 00 00 01 01 | 00 03 00 00 00 01 01 F4          ☺☺⌠  ☺☺ ♥   ☺☺⌠
000B71D0:    00 00 01 02 00 03 00 00 | 00 03 00 0B 72 72 01 03           ☺☻ ♥   ♥ ♂rr☺♥
000B71E0:    00 03 00 00 00 01 00 01 | 00 00 01 06 00 03 00 00         ♥   ☺ ☺  ☺♠ ♥
000B71F0:    00 01 00 02 00 00 01 0E | 00 02 00 00 00 07 00 0B         ☺ ☻  ☺♫ ☻    • ♂
000B7200:    72 78 01 11 00 04 00 00 | 00 01 00 00 00 08 01 12         rx☺◄ ♦   ☺   ◘☺↕
000B7210:    00 03 00 00 00 01 00 01 | 00 00 01 15 00 03 00 00         ♥   ☺ ☺  ☺§ ♥
000B7220:    00 01 00 03 00 00 01 16 | 00 03 00 00 00 01 01 F4         ☺ ♥  ☺▬ ♥    ☺☺⌠
000B7230:    00 00 01 17 00 04 00 00 | 00 01 00 0B 71 B0 01 1A          ☺↨ ♦   ☺ ♂q░☺→
000B7240:    00 05 00 00 00 01 00 0B | 72 80 01 1B 00 05 00 00         ♣   ☺ ♂rÇ☺← ♣
000B7250:    00 01 00 0B 72 88 01 1C | 00 03 00 00 00 01 00 01         ☺ ♂rê☺∟ ♥   ☺ ☺
000B7260:    00 00 01 28 00 03 00 00 | 00 01 00 02 00 00 00 00          ☺( ♥   ☺ ☻
000B7270:    00 00 00 08 00 08 00 08 | 57 41 4E 47 5A 02 00 00           ◘ ◘ ◘WANGZ☻
000B7280:    12 C0 00 00 00 04 00 00 | 12 C0 00 00 00 04 00 00         ↕└   ♦   ↕└   ♦
```

Figure 4.18: Hex Dump of the Image File Directory (IFD) from the end of image test2.tif in Figure 3.2

In this image, fifteen fields (or tags) are used to further define the IFD. These fields are color coded in the HEX data of Figure 4.18 and their corresponding data types and descriptions follows:

```
Tag 1 01 00 00 03 00 00 00 01 01 F4 00 00
Width, Short Data Type (16-bit unsigned integer), One Data Item, Width= Hex
01F4 = 500 pixels

Tag 2 01 01 00 03 00 00 00 01 01 F4 00 00
Height, Short Data Type (16-bit unsigned integer), One Data Item, Height= Hex
01F4 = 500 pixels

Tag 3 01 02 00 03 00 00 00 03 00 0B 72 72
Bits Per Sample, Short Data Type (16-bit unsigned integer), Three Data Items,
Offset=00 0B 72 72 points to the Hex values 00 08 00 08 00 08 which means the
image use three samples of 8-bits each.

Tag 4 01 03 00 03 00 00 00 01 00 01 00 00
Compression, Short Data Type (16-bit unsigned integer), One Data Item,
Compression Type=1 (no compression)
```

[illegible]
Photometric Interpretation, [illegible]), One Data
Item, value 2 meaning that [illegible]

Tag 6 01 0E 00 02 00 00 00 07 00 0B 72 78
Image Description, ASCII Data Type (8-bit NULL-terminated string), Seven Data Items.

Tag 7 01 11 00 04 00 00 00 01 00 00 00 08
Strip Offsets, Long Data Type (32-bit unsigned integer), One Data Item having value of 8.means that the image Raster Data begins at offset 8 in the image file. This is the position of the Red channel in the first pixel of an uncompressed RGB TIFF. Also, because there is only one data item (instead of an array of strip offsets) we know that there is only one strip of raster data in this image file.

Tag 8 01 12 00 03 00 00 00 01 00 01 00 00
Orientation, Short Data Type (16-bit unsigned integer), One Data Item with value of 1 meaning that the start of raster data corresponds to the upper left corner of the image.

Tag 9 01 15 00 03 00 00 00 01 00 03 00 00
Samples Per Pixel, Short Data Type (16-bit unsigned integer), One Data Item with value of 3 meaning that three samples are used per pixel. In Tag 3 above each sample is defined as having 8-bits.

Tag 10 01 16 00 03 00 00 00 01 01 F4 00 00
Rows Per Strip, Short Data Type (16-bit unsigned integer), One Data Item with value of 500 meaning that strips have 500 rows each. Because we know from Tag 2 that the height is 500 pixels, we know that there is only one strip of raster data.

Tag 11 01 17 00 04 00 00 00 01 00 0B 71 B0
Strip Byte Counts, Long Data Type (32-bit unsigned integer), One Data Item with value 750,000 meaning that the single strip of raster data uses 750,000 bytes. This is expected because the width of 500 and height of 500 yields 250,000 pixels, with each pixel requiring 3 bytes, resulting in a total of 750,000 bytes in TIFF file that uses no compression

Tag 12 01 1A 00 05 00 00 00 01 00 0B 72 80
XResolution, Rational Data Type (Two 32-bit unsigned integers), One Data Item pointing to the position 00 0B 72 80 in the file. At this position the numerator is Hex 12 C0 00 00 and the denominator is Hex 00 04 00 00, giving a rational number of Hex 04 B0 or 1200 decimal. This means that there are 1200 pixels per resolution unit (see tag 15) in X coordinate space.

Tag 13 01 1B 00 05 00 00 00 01 00 0B 72 88
YResolution, Rational Data Type (Two 32-bit unsigned integers), One Data Item pointing to the position 00 0B 72 88 in the file. At this position the numerator is Hex 12 C0 00 00 and the denominator is Hex 00 04 00 00, giving a rational number of Hex 04 B0 or 1200 decimal. This means that there are 1200 pixels per resolution unit (see Tag 15) in Y coordinate space.

Tag 14 01 1C 00 03 00 00 00 01 00 01 00 00
Planar Configuration, Short Data Type (16-bit unsigned integer), One Data Item with a value of 1 or Single Image Plane. A Planar Configuration tag is required for RGB TIFF images.

Tag 15 01 28 00 03 00 00 00 01 00 02 00 00
Resolution Unit, Short Data Type (16-bit unsigned integer), One Data Item with value of 2. This tag is used by Group 3 2d encoders. The data value is used with the XResolution (Tag 12) and YResolution (Tag 13) values. A data value of 2 is defined as the resolution in inches, so the resolution is in "pixels per inch." If the value was 3, then the resolution is in "pixels per centimeter." A data value of 1 is uncommon and does not specify an absolute unit of measurement. A value of 1 is used for images that may have a non-square aspect ratio, but no meaningful absolute dimensions. The drawback of ResolutionUnit=1

```
is that different applications will import the image at different sizes. Even
if the decision is quite arbitrary, it might be better to use dots (pixels) per
inch or dots (pixels) per centimeter, and pick XResolution and YResolution such
that the aspect ratio is correct.
```

Following the fifteen tags identified, the next four bytes specify the offset to the next IFD (if one exists). In the case of the data in Figure 4.18, the value of this offset is zero because there is no other IFD. Bytes that follow the IFDs contain data pointed to by IFD tags and may contain data too large to fit within the 12-byte IFD Tag size limitation. This data appears at the end of test2.tif (Figure 3.2) between offsets 0B7272 and 0B728F as illustrated by the **bold case**: hexadecimal values in Figure 4.19.

```
Offset       Hexadecimal Data                                              ASCII Data
000B7270:    00 00 00 08 00 08 00 08 | 57 41 4E 47 5A 02 00 00              ◘ ◘ ◘WANGZ☻
000B7280:    12 C0 00 00 00 04 00 00 | 12 C0 00 00 00 04 00 00            ↕└    ♦  ↕└   ♦
```

Figure 4.19: Hex Dump of data following the Image File Directory (IFD) from test2.tif

By further examining the structure of the IFD tags, we can examine this additional information. The likely candidate tag that would point to data beyond its 12-byte limitation is Tag 6: the Image Description

Tag 6 `01 0E 00 02 00 00 00 07 00 0B 72 78`

Image Description runs from `00 0B 72 78`

The first two bytes of any tag identify the tag type. In this case `01 0E` tells us this tag is the image description. The next two bytes contain the second field of the tag structure, which identify the data type of the tag contents. In this case `00 02` indicates the image description information is in ASCII format. The next four bytes (`00 00 00 07`) specify the number of items in the tag data; in this case there are seven items. The remaining four bytes specify the offset where the data is located within the file: `00 0B 72 78` to `00 0B 72 7E` which contains the bytes to represent the following ASCII string: WANGZ☻.

The bytes between offset 08 and offset 0B71B0 contain the image raster data. These are the bytes that make up the pixels we see on the screen when the image is displayed. The color channels in the TIFF file are organized as Red Green Blue (RGB).

Let us revisit the first part of image test2.tif (see Figure 3.2) to describe some of the methods employed for steganalysis (See Figure 4.20).

```
OFFSET      Hexadecimal Data                                       ASCII Data
00000000:   4D 4D 00 2A 00 0B 71 B8 | 6F 67 57 53 4F 3F 5B 5B      MM * ♂q┐ogWSO?[[
00000010:   4B 47 3B 37 53 4B 3F 4B | 3F 3F 4F 4F 4B 63 5F 5B      KG;7SK?K??OOKc_[
00000020:   6B 63 5F 6B 6F 6B 7F 87 | 83 7F 83 7B 77 7B 7B 77      kc_kok⌂çâ⌂â{w{{w
00000030:   77 6F 73 6F 6B 6B 5B 53 | 6B 57 53 6F 67 57 67 57      wosokk[SkWSogWgW
00000040:   3B 73 4F 3F 63 57 47 6B | 5F 57 77 73 6B 77 6F 67      ;sO?cWGk_Wwskwog
00000050:   8B 87 77 8B 7F 6B 6B 73 | 5B 7F 67 57 6F 6F 57 6B      ïçwï⌂kks[⌂gWooWk
00000060:   5B 4F 6B 67 5B 8B 97 7F | 3F 43 3F 6B 5B 5F 7F 67      [Okg[ïù⌂?C?k[_⌂g
00000070:   6B 7F 5B 5B 93 73 47 97 | 7B 47 6F 6F 4A 6B 67 5E      k⌂[[ôsGù{GooJkg^
00000080:   5F 56 52 67 66 6A 5B 4F | 56 6B 5F 63 5B 46 5A 6A      _VRgfj[OVk_c[FZj
00000090:   61 65 49 5D 51 35 39 41 | 61 60 5C 41 40 3C 35 34      aeI]Q59Aa`\A@<54
000000A0:   40 3D 38 38 49 3C 39 4D | 34 38 49 38 38 3D 3C 38      @=88I<9M48I88=<8
000000B0:   45 40 34 45 40 38 45 40 | 3C 45 44 38 41 3C 38 45      E@4E@8E@<ED8A<8E
000000C0:   34 34                                                  44
```

Figure 4.20: Hex Dump and ASCII data of the first 194 bytes of test2.tif from Figure 3.2

In our example we will look at the raster data from Figure 4.20 ranging from offset 08 to C1.

The color channels in the TIFF file are organized as Red Green Blue (RGB). The orientation of the pixels as they are stored in the file is from the upper left corner to the lower right corner of the image (see Figure 4.21). Not all image formats share this property. For example a BMP image will look identical on the screen but is stored quite differently in the physical file. Pixel data in BMP images are stored within a file with orientation following from bottom to top.

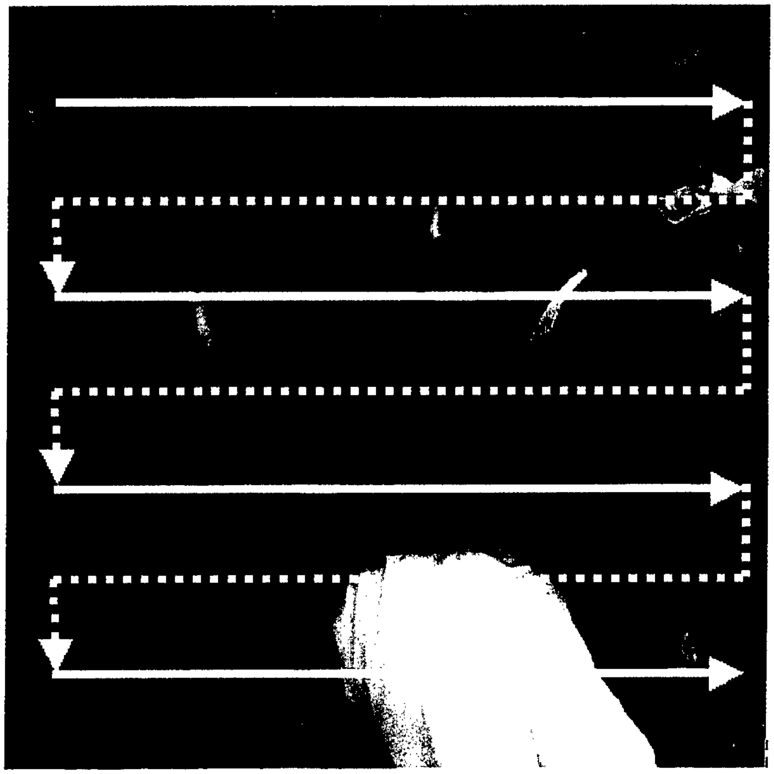


Figure 4.21: Typical orientation of the pixel storage within a TIFF image. (Revisiting test2.tif from Figure 3.2)

The first pixel in the image test2.tif (Figure 3.2) begins at offset 08 (following the header) and has the RGB byte values of 6F 67 57. Table 1 represents the hex and binary values of bytes, from offset 08 to C1, broken into separate bit planes. The table also indicates to which pixel and color channel a given byte belongs. (The color coding used in this table will be discussed later).

Table 1: Example of binary values of pixels and color channels of the data from Figure 3.2.

			Binary Values							
Pixel	Channel	Hex Value	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
1	R	6F	0	1	1	0	1	*1*	1	*1*
	G	67	0	1	1	*0*	0	1	1	*1*
	B	57	0	*1*	0	1	0	1	1	*1*
2	R	53	0	1	0	1	0	*0*	1	*1*
	G	4F	0	1	0	*0*	1	1	1	*1*
	B	3F	0	*0*	1	1	1	1	1	*1*

			Binary Values							
Pixel	Channel	Hex Value	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
3	R	5B	0	1	0	1	1	*0*	1	*1*
	G	5B	0	1	0	*1*	1	0	1	*1*
	B	4B	0	*1*	0	0	1	0	1	*1*
4	R	47	0	1	0	0	0	*1*	1	*1*
	G	3B	0	0	1	*1*	1	0	1	*1*
	B	37	0	*0*	1	1	0	1	1	*1*
5	R	53	0	1	0	1	0	*0*	1	*1*
	G	4B	0	1	0	*0*	1	0	1	*1*
	B	3F	0	*0*	1	1	1	1	1	*1*
6	R	4B	0	1	0	0	1	*0*	1	*1*
	G	3F	0	0	1	*1*	1	1	1	*1*
	B	3F	0	*0*	1	1	1	1	1	*1*
7	R	4F	0	1	0	0	1	*1*	1	*1*
	G	4F	0	1	0	*0*	1	1	1	*1*
	B	4B	0	*1*	0	0	1	0	1	*1*
8	R	63	0	1	1	0	0	*0*	1	*1*
	G	5F	0	1	0	*1*	1	1	1	*1*
	B	5B	0	*1*	0	1	1	0	1	*1*
9	R	6B	0	1	1	0	1	*0*	1	*1*
	G	63	0	1	1	*0*	0	0	1	*1*
	B	5F	0	*1*	0	1	1	1	1	*1*
10	R	6B	0	1	1	0	1	*0*	1	*1*
	G	6F	0	1	1	*0*	1	1	1	*1*
	B	6B	0	*1*	1	0	1	0	1	*1*
11	R	7F	0	1	1	1	1	*1*	1	*1*
	G	87	1	0	0	*0*	0	1	1	*1*
	B	83	1	*0*	0	0	0	0	1	*1*
12	R	7F	0	1	1	1	1	*1*	1	*1*
	G	83	1	0	0	*0*	0	0	1	*1*
	B	7B	0	*1*	1	1	1	0	1	*1*
13	R	77	0	1	1	1	0	*1*	1	*1*
	G	7B	0	1	1	*1*	1	0	1	*1*
	B	7B	0	*1*	1	1	1	0	1	*1*
14	R	77	0	1	1	1	0	*1*	1	*1*
	G	77	0	1	1	*1*	0	1	1	*1*
	B	6F	0	*1*	1	0	1	1	1	*1*
15	R	73	0	1	1	1	0	*0*	1	*1*
	G	6F	0	1	1	*0*	1	1	1	*1*
	B	6B	0	*1*	1	0	1	0	1	*1*
16	R	6B	0	1	1	0	1	*0*	1	*1*
	G	5B	0	1	0	*1*	1	0	1	*1*
	B	53	0	*1*	0	1	0	0	1	*1*
17	R	6B	0	1	1	0	1	*0*	1	*1*
	G	57	0	1	0	*1*	0	1	1	*1*
	B	53	0	*1*	0	1	0	0	1	*1*
18	R	6F	0	1	1	0	1	*1*	1	*1*

			Binary Values							
Pixel	Channel	Hex Value	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
	G	67	0	1	1	*0*	0	1	1	*1*
	B	57	0	*1*	0	1	0	1	1	*1*
	R	67	0	1	1	0	0	*1*	1	*1*
19	G	57	0	1	0	*1*	0	1	1	*1*
	B	3B	0	*0*	1	1	1	0	1	*1*
	R	73	0	1	1	1	0	*0*	1	*1*
20	G	4F	0	1	0	*0*	1	1	1	*1*
	B	3F	0	*0*	1	1	1	1	1	*1*
	R	63	0	1	1	0	0	*0*	1	*1*
21	G	57	0	1	0	*1*	0	1	1	*1*
	B	47	0	*1*	0	0	0	1	1	*1*
	R	6B	0	1	1	0	1	*0*	1	*1*
22	G	5F	0	1	0	*1*	1	1	1	*1*
	B	57	0	*1*	0	1	0	1	1	*1*
	R	77	0	1	1	1	0	*1*	1	*1*
23	G	73	0	1	1	*1*	0	0	1	*1*
	B	6B	0	*1*	1	0	1	0	1	*1*
	R	77	0	1	1	1	0	*1*	1	*1*
24	G	6F	0	1	1	*0*	1	1	1	*1*
	B	67	0	*1*	1	0	0	1	1	*1*
	R	8B	1	0	0	0	1	*0*	1	*1*
25	G	87	1	0	0	*0*	0	1	1	*1*
	B	77	0	*1*	1	1	0	1	1	*1*
	R	8B	1	0	0	0	1	*0*	1	*1*
26	G	7F	0	1	1	*1*	1	1	1	*1*
	B	6B	0	*1*	1	0	1	0	1	*1*
	R	6B	0	1	1	0	1	*0*	1	*1*
27	G	73	0	1	1	*1*	0	0	1	*1*
	B	5B	0	*1*	0	1	1	0	1	*1*
	R	7F	0	1	1	1	1	*1*	1	*1*
28	G	67	0	1	1	*0*	0	1	1	*1*
	B	57	0	*1*	0	1	0	1	1	*1*
	R	6F	0	1	1	0	1	*1*	1	*1*
29	G	6F	0	1	1	*0*	1	1	1	*1*
	B	57	0	*1*	0	1	0	1	1	*1*
	R	6B	0	1	1	0	1	*0*	1	*1*
30	G	5B	0	1	0	*1*	1	0	1	*1*
	B	4F	0	*1*	0	0	1	1	1	*1*
	R	6B	0	1	1	0	1	*0*	1	*1*
31	G	67	0	1	1	*0*	0	1	1	*1*
	B	5B	0	*1*	0	1	1	0	1	*1*
	R	8B	1	0	0	0	1	*0*	1	*1*
32	G	97	1	0	0	*1*	0	1	1	*1*
	B	7F	0	*1*	1	1	1	1	1	*1*
33	R	3F	0	0	1	1	1	*1*	1	*1*
	G	43	0	1	0	*0*	0	0	1	*1*

			Binary Values							
Pixel	Channel	Hex Value	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
	B	3F	0	0	1	1	1	1	1	1
	R	6B	0	1	1	0	1	0	1	1
34	G	5B	0	1	0	1	1	0	1	1
	B	5F	0	1	0	1	1	1	1	1
	R	7F	0	1	1	1	1	1	1	1
35	G	67	0	1	1	0	0	1	1	1
	B	6B	0	1	1	0	1	0	1	1
	R	7F	0	1	1	1	1	1	1	1
36	G	5B	0	1	0	1	1	0	1	1
	B	5B	0	1	0	1	1	0	1	1
	R	93	1	0	0	1	0	0	1	1
37	G	73	0	1	1	1	0	0	1	1
	B	47	0	1	0	0	0	1	1	1
	R	97	1	0	0	1	0	1	1	1
38	G	7B	0	1	1	1	1	0	1	1
	B	47	0	1	0	0	0	1	1	1
	R	6F	0	1	1	0	1	1	1	1
39	G	6F	0	1	1	0	1	1	1	1
	B	4A	0	1	0	0	1	0	1	0
	R	6B	0	1	1	0	1	0	1	1
40	G	67	0	1	1	0	0	1	1	1
	B	5E	0	1	0	1	1	1	1	0
	R	5F	0	1	0	1	1	1	1	1
41	G	56	0	1	0	1	0	1	1	0
	B	52	0	1	0	1	0	0	1	0
	R	67	0	1	1	0	0	1	1	1
42	G	66	0	1	1	0	0	1	1	0
	B	6A	0	1	1	0	1	0	1	0
	R	5B	0	1	0	1	1	0	1	1
43	G	4F	0	1	0	0	1	1	1	1
	B	56	0	1	0	1	0	1	1	0
	R	6B	0	1	1	0	1	0	1	1
44	G	5F	0	1	0	1	1	1	1	1
	B	63	0	1	1	0	0	0	1	1
	R	5B	0	1	0	1	1	0	1	1
45	G	46	0	1	0	0	0	1	1	0
	B	5A	0	1	0	1	1	0	1	0
	R	6A	0	1	1	0	1	0	1	0
46	G	61	0	1	1	0	0	0	0	1
	B	65	0	1	1	0	0	1	0	1
	R	49	0	1	0	0	1	0	0	1
47	G	5D	0	1	0	1	1	1	0	1
	B	51	0	1	0	1	0	0	0	1
	R	35	0	0	1	1	0	1	0	1
48	G	39	0	0	1	1	1	0	0	1
	B	41	0	1	0	0	0	0	0	1

			Binary Values							
Pixel	Channel	Hex Value	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
	R	61	0	1	1	0	0	*0*	0	*1*
49	G	60	0	1	1	*0*	0	0	0	*0*
	B	5C	0	*1*	0	1	1	1	0	*0*
	R	41	0	1	0	0	0	*0*	0	*1*
50	G	40	0	1	0	*0*	0	0	0	*0*
	B	3C	0	*0*	1	1	1	1	0	*0*
	R	35	0	0	1	1	0	*1*	0	*1*
51	G	34	0	0	1	*1*	0	1	0	*0*
	B	40	0	*1*	0	0	0	0	0	*0*
	R	3D	0	0	1	1	1	*1*	0	*1*
52	G	38	0	0	1	*1*	1	0	0	*0*
	B	38	0	*0*	1	1	1	0	0	*0*
	R	49	0	1	0	0	1	*0*	0	*1*
53	G	3C	0	0	1	*1*	1	1	0	*0*
	B	39	0	*0*	1	1	1	0	0	*1*
	R	4D	0	1	0	0	1	*1*	0	*1*
54	G	34	0	0	1	*1*	0	1	0	*0*
	B	38	0	*0*	1	1	1	0	0	*0*
	R	49	0	1	0	0	1	*0*	0	*1*
55	G	38	0	0	1	*1*	1	0	0	*0*
	B	38	0	*0*	1	1	1	0	0	*0*
	R	3D	0	0	1	1	1	*1*	0	*1*
56	G	3C	0	0	1	*1*	1	1	0	*0*
	B	38	0	*0*	1	1	1	0	0	*0*
	R	45	0	1	0	0	0	*1*	0	*1*
57	G	40	0	1	0	*0*	0	0	0	*0*
	B	34	0	*0*	1	1	0	1	0	*0*
	R	45	0	1	0	0	0	*1*	0	*1*
58	G	40	0	1	0	*0*	0	0	0	*0*
	B	38	0	*0*	1	1	1	0	0	*0*
	R	45	0	1	0	0	0	*1*	0	*1*
59	G	40	0	1	0	*0*	0	0	0	*0*
	B	3C	0	*0*	1	1	1	1	0	*0*
	R	45	0	1	0	0	0	*1*	0	*1*
60	G	44	0	1	0	*0*	0	1	0	*0*
	B	38	0	*0*	1	1	1	0	0	*0*
	R	41	0	1	0	0	0	*0*	0	*1*
61	G	3C	0	0	1	*1*	1	1	0	*0*
	B	38	0	*0*	1	1	1	0	0	*0*
	R	45	0	1	0	0	0	*1*	0	*1*
62	G	34	0	0	1	*1*	0	1	0	*0*
	B	34	0	*0*	1	1	0	1	0	*0*

Our main goal is to determine whether a given image contains embedded information and, if it does, to possibly extract that information. If we assume that information is hidden in the pixel data (the spatial domain) and not in some transform coefficients (the frequency domain), then we are interested in analyzing a subset of the pixel data. The subset of interest is that which contains the hidden information. One of the most common subsets used in hiding techniques, as discussed above, is LSB substitution. The LSBs (the higher order the bit used, the easier the detection is) will be overwritten with the hidden information using this particular hiding technique.

Example---Pixel Walk LSB Insertion Technique:

We use as a detailed example yet another method of LSB steganography. This method only works for color images. We use this example in detail to discuss how steganography might be detected. (Note that the Kurak-McHugh method swaps entire bit planes of the embedded image with the cover image and hence is trivial to detect --- we will come back to this later.) The pixel walk LSB insertion technique is a cross between the Kurak-McHugh method and the NRL method. The information that is to be hidden in the image can be separated into a stream of individual bits. The simplest pixel walk LSB insertion technique hides the information in the pixel data as follows: one bit from the stream is inserted into the LSB of the red channel of the first pixel, replacing the original value but changing the byte value by no more than 1. The next bit from the stream is written to the LSB of the green channel of the first pixel. This is followed by writing the next bit from the stream to the blue channel of the first pixel, which is then followed by moving on to the next pixel and repeating the process until the bit stream is empty. Figure 4.22 provides an illustration of LSB encoding of the binary value for the letter H in the LSBs of three pixels.

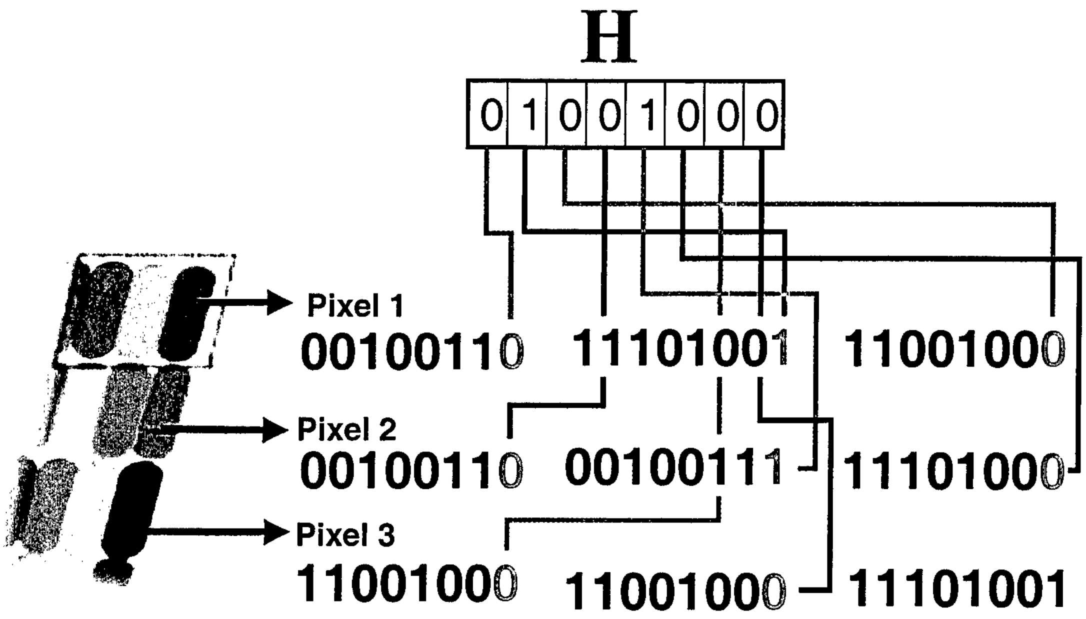


Figure 4.22: Encoding the binary value for ‘H’ into the LSBs of three pixels.

Let us further describe some concepts used in selecting the subset of pixel data of interest. *Offset* describes how many bytes to ignore from the start of pixel data before beginning the process. If *offset=2*, then we ignore the first two bytes of pixel data and begin the process on the third byte (the B byte of pixel 1). *Skip* describes how many bytes to ignore (or skip) between bytes of interest. If *offset=1* and *skip=1*, then we start the process on the second byte (G byte of pixel 1), ignore the third byte, process the fourth byte (R byte of pixel 2), ignore the fifth, and so on to the end of pixel data following this same pattern. We also use the term *bit plane* or *bit level* to describe which bit to take from every byte of interest. For instance, if *offset=0*, *skip=2*, and *bit level=1*, then we start on the first byte and process the first bit of every third byte. This is equivalent to saying that we will take the LSB of every byte in the red channel of the image.

The following four color-coded examples illustrate the terms *offset*, *skip*, and *bit level* by showing how the terms are used in practice. By comparing the values shown below to their representation in Table 1, the meaning of each term should become clear.

Offset = 0, skip = 0, bit level = 1. (All LSBs)

Offset = 0, skip = 2, bit level = 3. (All red 3rd order bits)

Offset = 1, skip = 2, bit level = 5. (All green 5th order bits)

Offset = 2, skip = 2, bit level = 7. (All blue 7th order bits)

If we were going to analyze all the LSBs of an image, then we would analyze the subset of pixel data with *offset=0*, *skip=0*, and *bit level=1*, which corresponds to the orange-coded bits in the pixel data in the above table. That subset should contain all hidden information that was embedded in the image in the pixel walk manner described above. After the subset has been obtained it is just a series of bits that must be treated further to arrive at the original information as it was before it was hidden in the image. If we assume the original information is a series of bytes and that the bits were hidden from most-significant-bit to least-significant-bit, then we simply divide the captured bits into a series of consecutive groups of eight bits, allowing the first bit in each group to be the most significant bit. We then arrive at a series of bytes that is identical to the original information before it was hidden. This process is described below.

The simplest pixel walk LSB insertion technique hides one bit in the LSB of every byte of pixel data, starting at the first pixel of an image and continuing until the end of the data is reached. In the example from Table 1 above is from the image data for test2.fig (Figure 3.2). The bits of interest (the LSBs in the target image) are those that appear in orange. Here they are as a subset, appearing in consecutive order:

```
111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111110110100100110111100011111111
100100100100101100100100100100100100100100100...
```

Now we divide this series into 8-bit groups that we will treat as bytes:

```
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11110110 10010011
01111000 11111111 10010010 01001011 00100100 10010010 01001001 00...
```

As bytes, we now have a stream of data with hex values FF FF FF FF FF FF FF FF FF FF FF FF FF FF F6 93 78 FF 92 4B 24 92 49 …

Now the task becomes one of discerning meaning. From what has been shown so far it is not clear whether this data means anything. It certainly does not appear to be ASCII text as most of the values do not fall within the ASCII range (standard ASCII characters have the higher order bit set to zero). However, it does not appear random either because of the great repetition of values at the start of the data. Even the casual observer would wonder why there are so many consecutive ones. But it is not uncommon for an image to have substantial repetition of consecutive color values. However, a look at the original color values from which this subset was taken shows that not to be the case: the color values are quite varied. Instead, examination of this subset seems to indicate deliberate placement of odd values instead of even values at the start of image data. There is no guarantee yet that this is an indication of information hiding, but it is unlikely that this pattern is natural. Therefore the pattern of bits raises suspicion and this image becomes suspect. We used the pixel walk LSB insertion method as an illustrative example. As it turns out, Figure 3.2 is a stego image.[6] CSIS was not told this ahead of time. The steganographic method used was Kurak-McHugh, where the 2-LSBs of test1.tif replaced the 2-LSBs of the cover image embedded image (an image identical to test2.tif in the six high order bits) to form the stego image test2.tif. Therefore, the above steganalysis detected that the least bit plane had been manipulated. Of course other similar tests can determine more.

Methods Employed for Image Analysis

The following provides explanation of the methods we used in investigating the NRL test images:

Observing Histogram Anomalies: As a crude first step at analyzing a given file, we may use this technique to get a statistical illustration of a file's distribution of data. For every byte in a file, we enter the byte value (from 0 to 255) into a histogram. When complete, the frequency of occurrence of each value is known. By examining this histogram, statistical anomalies or trends may be identified. This evaluation may also suggest an appropriate set of more definitive follow up analysis and provides input for other tests.

Image Bit-Plane Separation: The idea behind this technique is that hiding methods embed information in only a subset of available data. In images, one potential subset is a single bit plane in the pixel data (e.g., using only the 2nd order bit in every byte). This analytical technique involves separating the image into planes and relying on the HVS to detect patterns when viewing a representation of a single plane. We produce a new image that reduces the original image to only one of its bit planes. First, a bit level is selected so that only one bit in every byte is used. If the selected bit in the original byte is a 0, the byte value in the new image will be a 0. If the selected bit in the original byte is a 1, the byte value in the new image will be 255. This makes patterns more visible.

[6] test1.tif and test2.tif were sent by NRL to CSIS to test the testers!

The output can be examined visually to determine if lower bit planes correlate with upper bit planes, or if lower bit planes contain some uncorrelated pattern that may indicate embedded information in the lower bit levels.

Data Subset Reconstruction: The goal of this process is to select a subset of data within a suspect file and then to reconstruct the original hidden data from this subset. Of course, we will only succeed if we select the exact subset in which information was embedded because the entire subset of data is used in the reconstruction. The output of this process is a new file containing a concatenation of the bits contained in the extracted subset. In extracting the subset, we select up to eight bit levels, although if all eight were selected the result would be an identical byte-for-byte copy of the original file. It is more practical to select only the bits where information is likely to have been hidden. (Using high order bits is a sure way of being detected.) We may also use a byte offset from the start of the original file to avoid extracting bits from the file header in file types that have headers where information is unlikely to be hidden. If, for example, the lower two bits of every byte were used to store hidden information, then we could select these two bit levels and then proceed through the original file byte by byte, extracting only the lower two bits from each original byte and placing them in a new byte. Each new byte would be constructed from four original bytes using only the two lower bits from each byte. The new file would be created by concatenation of these new bytes. If the information was embedded in sequence without further encryption, then this technique may extract the embedded message.

Detection of General LSB Manipulation: When we process an image file with the general LSB detection technique, we skip over any file header information and begin loading values into a histogram after entering the pixel data. For every pixel, the red, green, and blue channel values are summed. The value of the sum modulo 256 (resultant range 0 to 255) is entered into the histogram. When all pixels have been processed in this way, the values in the histogram are treated as observations in a standard chi-squared (X^2) test. The histogram is broken into 128 pairs of adjacent odd and even values, with each pair referred to as a pair of values (POV). The first POV contains histogram elements 0 and 1, and the 128th POV contains histogram elements 254 and 255. Overall, the values within individual POVs are expected to be different in images that do not have manipulated LSBs (i.e., do not have information hidden in their LSBs). If, however, the POVs in the histogram are significantly similar, the image is suspected to have been steganographically altered. This technique is still experimental but its reliability remains promising.

Detection of Fixed-Pattern LSB Manipulation: This technique differs from the general LSB manipulation detection technique only in the way that it loads the histogram. Because specific hiding/embedding algorithms change only certain LSBs during the embedding process, only those byte values are entered into the histogram. For example, a possible embedding technique might hide a single bit in every third byte of pixel data. Therefore, the value of every third byte is entered into the histogram for all the image data. When all pixels have been processed in this way, the values in the histogram are subjected to a X^2 test in exactly the same way as in the general LSB test. Again, overall similarity of the values within individual POVs sets the level of suspicion, as determined by the output of the X^2 test.

Underlying X^2 Test: The X^2 statistical test relies on the idea that manipulation of image data to hide information will result in a change from uncorrelated pairs of values to correlated (or similar) pairs of values in the histograms produced as described above.

Other X^2 Testing: Analysis of non-image files can be accomplished with a process modified from the techniques mentioned above. The most important step is providing for a way to limit testing to data where information is expected to have been hidden. If we suspect that only a specific subset of bytes is being used to carry the hidden data, then only the values in those bytes should be entered into the histogram. Using an offset from the beginning of the file and allowing for bytes to be skipped within the file makes this possible.

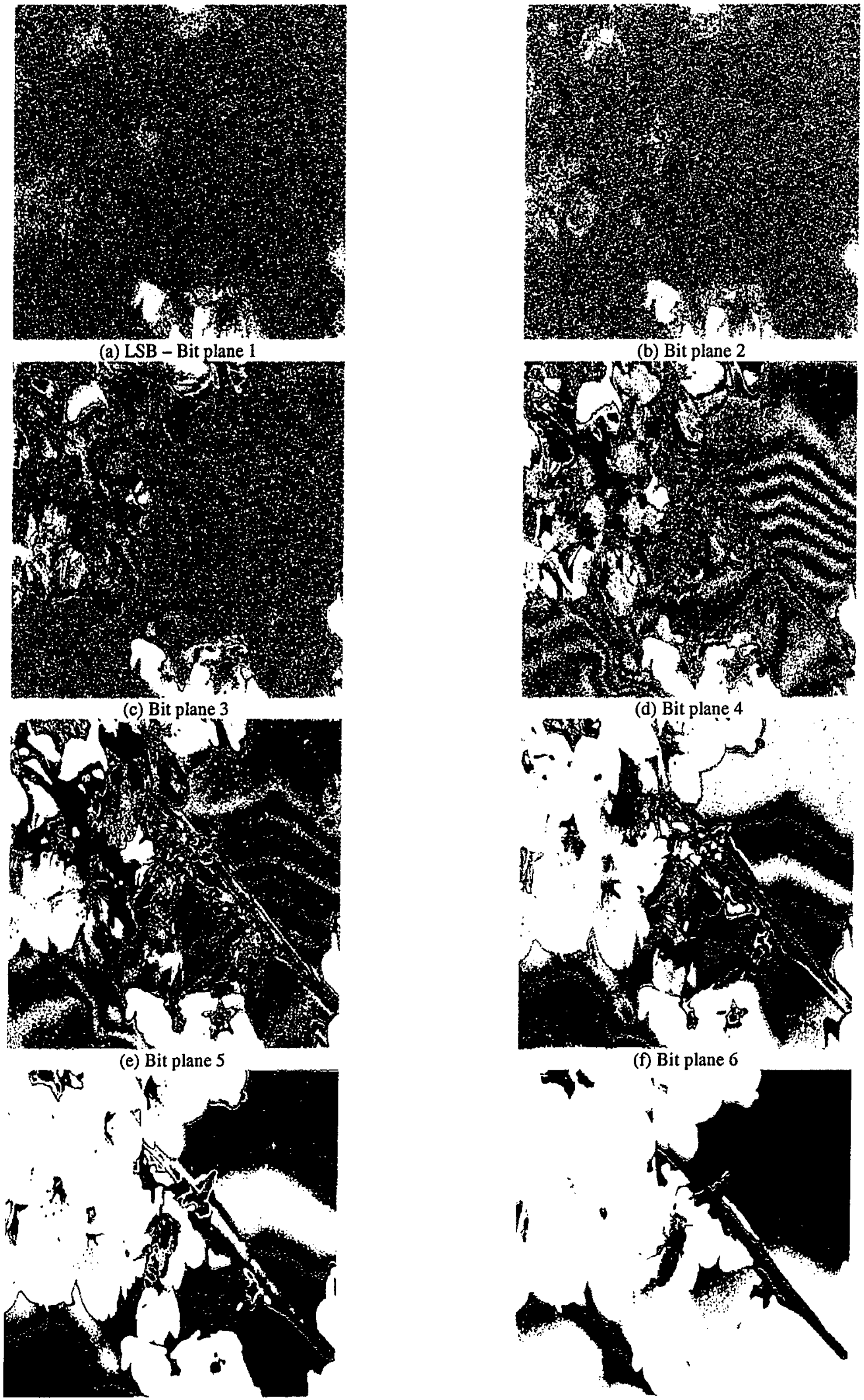
5 Experimental Results

The analytic tests employed by CSIS upon receipt of the images from NRL were as follows:

1. Review the distribution of bits in each of the 8 levels of each color plane.
2. Review the statistical properties of the bits in the overall image and then within the bit planes. Methods include those mentioned above.

These two tests alone will direct the investigators to further actions. These tests can be used to discover the randomness of data in each of the bit planes. The existence of random data my lead one to expect encrypted data. Having a high number of unique colors with respect to the overall size of the images may be the result of embedded information. Also having a large number of colors differing by one bit per color channel may be the result of least significant bit (LSB) manipulation to hide data.

The images in Figure 5.23a,...,h show the eight bit planes of test1.tif (see Figure 3.1) and the images in Figure 5.24a,...h show the eight bit planes from test2.tif (see Figure 3.2). The colors of each level are enhanced for better observation. We can clearly see that the upper two bit planes (Figure 5.23g and h) are embedded into the lowest two bits of the image test2.tif (Figure 5.24g and h). This can also be detected by a statistical test such as the discrete Laplacian [5].



(a) LSB – Bit plane 1

(b) Bit plane 2

(c) Bit plane 3

(d) Bit plane 4

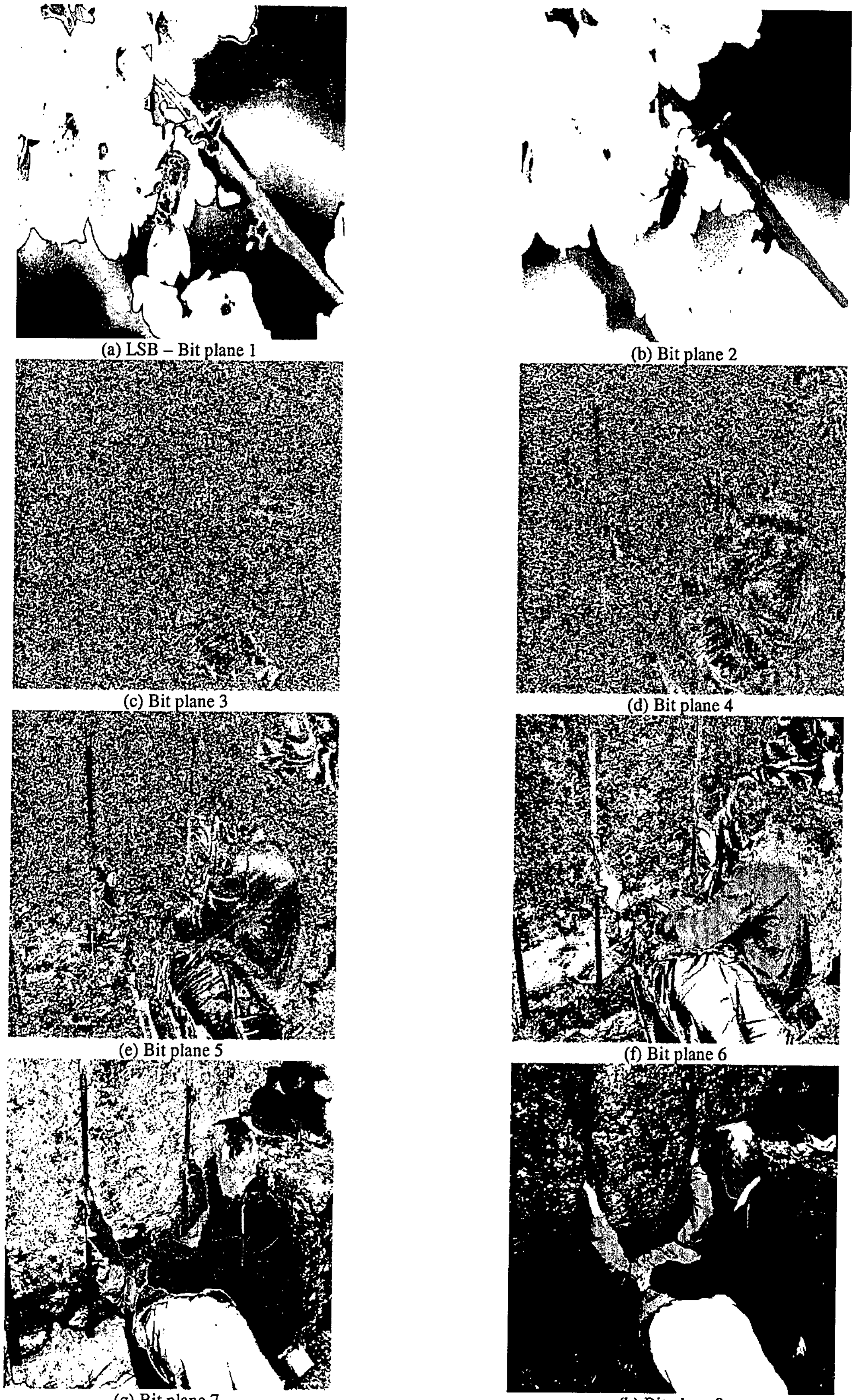
(e) Bit plane 5

(f) Bit plane 6

(g) Bit plane 7

(h) Bit plane 8

Figure 5.23: Eight bit planes of test1.tif



(a) LSB – Bit plane 1

(b) Bit plane 2

(c) Bit plane 3

(d) Bit plane 4

(e) Bit plane 5

(f) Bit plane 6

(g) Bit plane 7

(h) Bit plane 8

Figure 5.24: Eight bit planes from test2.tif

Running further experiments in measuring the relationship between colors and near pixels, CSIS found that the test1.tif (Figure 3.1) image seems to have been processed using a steganography tool that employs a LSB embedding technique. (In fact it was processed using the NRL method but we are suspicious that this was a false positive, with respect to the NRL method, perhaps due to some manipulation prior to NRL encoding any message. This suspicion is given weight due to the fact that CSIS could not detect the NRL method in any of the other test images.) We have not identified the tool, but the pattern is consistent with techniques similar to S-Tools when processing 24-bit images. It is possible that a 24-bit BMP file can be processed with a steganography tool and then converted to a lossless format such as the 24-bit TIFF files provided by NRL.

According to CSIS the images from Figure 3.1 and Figure 3.2 used trivial and easily detectable methods of data encoding (Figure 3.1 had no steganography used upon it, whereas Figure 3.2 results from the Kurak-McHugh method. Therefore the result for Figure 3.1 was a false positive.) CSIS detected no steganography, using their various statistical methods for Figure 3.3 through Figure 3.16, with the exception of Figure 3.15. When one views Figure 3.15 in its normal size one can easily see that something is amiss. That is there are bright spots in it. This is because instead of just modifying the R byte we modified all three colors to have the same value as the R byte, and thus the steganography becomes obvious (one should not even call it steganography in that case). If R is a high value, and all three colors are set to the same value the pixel then appears as a bright spot. This is why we only modify the R byte.

The tables below describe the various test images (#characters is the size of the embedded message):

Image	TIFF	Size	Detect	Technique	#Characters
test1.tif	color	500x500	?	NRL method	99
test2.tif	color	500x500	Yes	*KurakMcHugh*	2LSB-image
test3.tif	color	500x500	No	none	none
test4.tif	color	500x500	No	NRL method	64
test5.tif	greyscale	500x500	No	none	none
test6.tif	color	500x500	No	NRL method	208
test7.tif	greyscale	500x500	No	NRL method	95
test8.tif	greyscale	792x528	No	none	none
test9.tif	greyscale	500x500	No	none	none
test10.tif	greyscale	500x500	No	NRL method	74
test11.tif	greyscale	720x576	No	none	none
test12.tif	greyscale	776x552	No	none	none
test13.tif	color	500x500	No	none	none
test14.tif	greyscale	500x500	No	NRL method	6
test15.tif	color	576x720	HVS	modified NRL	85
test16.tif	greyscale	500x500	No	NRL method	53